Øving 11 - Numerisk integrasjon - LF

Obligatoriske oppgaver

The code for these exercises is given below.

$\boxed{1}$ For $n = 2503$ we get an error of $\approx 10^{-7}$.

$\boxed{2}$ For $n = 12$ we get an error of $\approx 10^{-14}$.

$\boxed{3}$ For $n = 512$ we get an error of $\approx 10^{-8}$. Why this bad approximation (high number of n)? The Chebychev points tend to lie closer to the boundary points -1 and 1 than towards the middle of the interval. The function

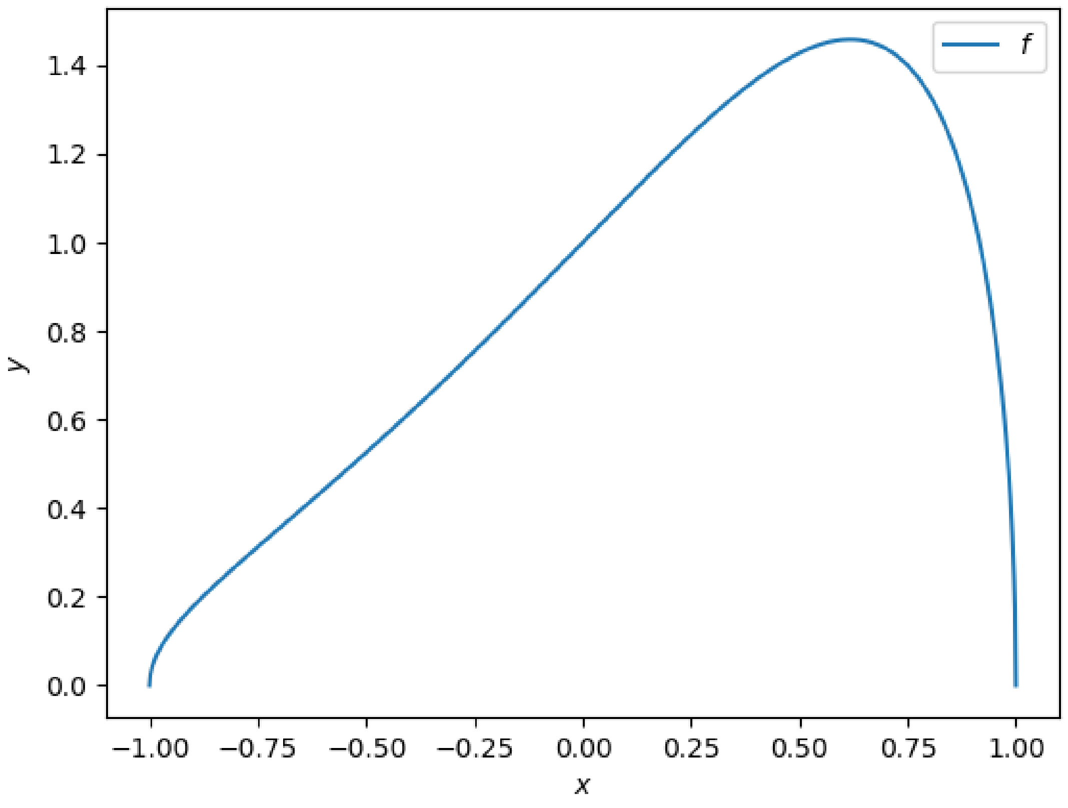
$$f(x) = e^x\sqrt{1-x^2}$$

is going very steeply towards 0 in the endpoints, while most of the mass is in the middle. The Chebychev points doesn't catch this information very well, and therefore the approximation is bad. A figure of f is given below.

$\boxed{4}$ We get an error of $\approx 10^{-12}$. Comparing to exercise 1 and 2, this is a very good method. It's really fast (one iteration), and gives a better result than the trapezoid rule for $n = 2503$ and comparable results to the Clenshaw-Curtis rule.

$\boxed{\text{Code:}}$ **Python:**

```
import numpy as np

#Our functions
def f(x):
    return np.exp(x)

def g(x):
    return np.multiply(np.exp(x), np.sqrt(1-np.square(x)))



#Our methods
def composite_trapezoid(a,b,f,n):
    #implementasjon av sammensatt trapesregel
    h = (b-a)/n
    sum = h/2*(f(a)+f(b))
    for i in range(1,n):
        sum = sum + h*f(a+i*h)
    return sum


def clenshaw_curtis(f,n):
    #Returns integral of f from -1 to 1
    #using n weights.

    if not (n %2 ==0):
        print('The number n is not even!')
```



Figur 1: Plot of f

```
    return

#Chebychev ekstremalpunktgitter
tmp = np.linspace(0,n−1,n)
x = np.cos( tmp *np.pi/(n−1))

#making of the weights
w = np.zeros(len(x))
w[0] = 1/((n−1)**2)
w[−1] = w[0]

for i in range(2,n):

    #Numpy array from 1 to (n−2)/2
    piste = np.linspace(1,int((n−2)/2), int((n−2)/2))

    #Arguments to be multiplied
    arg_1 = 2*np.power(4*np.square(piste)−1,−1)
    arg_2 = np.cos(2*np.pi*piste*(i−1)/(n−1))

    #The weights
    w[i−1] = 2.0*(1−np.sum( np.multiply(arg_1, arg_2)))/(n−1)


#The numerical integral.
```

```
    return np.dot(f(x),w)

def gauss_lobatto(f,a,b):
    #Returns integral of f from a to b

    #For the interval [-1,1] we have the following points and weights
    c_1 = np.sqrt(5.0/11 -2.0/11*np.sqrt(5.0/3))
    c_2 = np.sqrt(5.0/11 + 2.0/11*np.sqrt(5.0/3))


    x = np.array([0, c_1,-c_1, c_2, -c_2, 1,-1])

    #Skalering tilpasset intervallet [a,b]
    x = (b-a)*x/2 + (a+b)/2

    #The corresponding weights
    k_1 = (124+7*np.sqrt(15))/350
    k_2 = (124-7*np.sqrt(15))/350

    w = [256/525, k_1, k_1, k_2, k_2, 1/21, 1/21]

    return np.dot(f(x),w)

if __name__ == "__main__":

    #Feiltoleranse
    tol = 10.0**-10

    #####################################
    #Oppgave 1)

    print('-----------\n')
    print('Oppgave 1 (Sammensatt trapesregel): \n')

    #Feilen
    d = 1

    #Iterasjonen
    n = 2

    #Nyeste og forrige estimat
    new = old = composite_trapezoid(-1,1,f,n)

    #dersom det   velge st rre n ikke gir noe s rlig bedre estimat
    #enn toleransen, er vi ganske n r eksakt l sning
    while d>tol:
        n += 1
        new = composite_trapezoid(-1,1,f,n)
        d = np.abs(new-old)
        old = new
    #Noyaktig svar
    I = np.exp(1.0)-np.exp(-1.0)

    #Konkluderinger
    print("Resultat:",new)
    print("Eksakt svar:",I)
    print("Feil i svaret:",I-new)
    print("n =",n)


    #####################################
```

```
#Oppgave 2)

print('-----------\n')
print('Oppgave 2 (Clenshaw-Curtis): \n')

#Feilen
d = 1

#Iterasjonen
n = 2

#Nyeste og forrige estimat
new = old = clenshaw_curtis(f,n)

#Mens feilen er for stor
while d>tol:
    n +=2
    new = clenshaw_curtis(f,n)
    d = np.abs(new-old)
    old = new

#Noyaktig svar
I_1 = np.exp(1.0)-np.exp(-1.0)

#Konkluderinger
print("Resultat:",new)
print("Eksakt svar:",I_1)
print("Feil i svaret:",I_1-new)
print("n =",n)

#####################################
#Oppgave 3)

print('\n-----------\n')
print('Oppgave 3 (Clenshaw-Curtis): \n')

#Feilen
d = 1

#Iterasjonen
n = 2

#Nyeste og forrige estimat
new = old = clenshaw_curtis(g,n)

#Mens feilen er for stor
while d>tol:
    n+=2
    new = clenshaw_curtis(g,n)
    d = np.abs(new-old)
    old = new

#Noyaktig svar
I_2 = 1.7754996892121809469

#Konkluderinger
print("Resultat:",new)
print("Eksakt svar:",I_2)
print("Feil i svaret:",I_2-new)
print("n =",n)
```

```
    ####################################
    #Oppgave 4)

    print('\n------------\n')
    print('Oppgave 4 (Gauss-Lobatto): \n')

    #Numerisk integral
    b = gauss_lobatto(f,-1,1)

    #Konkluderinger
    print("Resultat:", b)
    print("Eksakt svar:" , I_1)
    print('Feil i svaret:', b-I_1)

    print('\n------------\n')
```

We get the following plot:

Anbefalte oppgaver

The code for these exercises is given below.

$\boxed{1}$ We get

$$\int_{-1}^{1} \frac{e^x}{\sqrt{1-x^2}} dx \approx 3.4865363609$$

where the true value is approximately 3.97746. Quite big error!

$\boxed{2}$ To approximate

$$\int_0^1 \ln x dx, \tag{1}$$

we need to be careful, as $\lim_{x\searrow 0} \ln x = -\infty$. Thus, in the code, we use the composite trapezoidal rule with n subintervals to estimate the integral

$$\int_{1/n}^{1} \ln(x) dx,$$

as this integral converges towards (1). We are able to reach an error of $\approx 10^{-3}$ after 2800 iterations with this method.

$\boxed{\text{Code:}}$ **Python:**

```
import numpy as np

#Our functions
def f(x):
    return np.exp(x)

def g(x):
    return np.multiply(np.exp(x), np.sqrt(1-np.square(x)))
```

```
#Our methods
def composite_trapezoid(a,b,f,n):
    #implementasjon av sammensatt trapesregel
    h = (b-a)/n
    sum = h/2*(f(a)+f(b))
    for i in range(1,n):
        sum = sum + h*f(a+i*h)
    return sum


def clenshaw_curtis(f,n):
    #Returns integral of f from -1 to 1
    #using n weights.

    if not (n %2 ==0):
        print('The number n is not even!')
        return

    #Chebychev ekstremalpunktgitter
    tmp = np.linspace(0,n-1,n)
    x = np.cos( tmp *np.pi/(n-1))

    #making of the weights
    w = np.zeros(len(x))
    w[0] = 1/((n-1)**2)
    w[-1] = w[0]

    for i in range(2,n):

        #Numpy array from 1 to (n-2)/2
        piste = np.linspace(1,int((n-2)/2), int((n-2)/2))

        #Arguments to be multiplied
        arg_1 = 2*np.power(4*np.square(piste)-1,-1)
        arg_2 = np.cos(2*np.pi*piste*(i-1)/(n-1))

        #The weights
        w[i-1] = 2.0*(1-np.sum( np.multiply(arg_1, arg_2)))/(n-1)


    #The numerical integral.
    return np.dot(f(x),w)

def gauss_lobatto(f,a,b):
    #Returns integral of f from a to b

    #For the interval [-1,1] we have the following points and weights
    c_1 = np.sqrt(5.0/11 -2.0/11*np.sqrt(5.0/3))
    c_2 = np.sqrt(5.0/11 + 2.0/11*np.sqrt(5.0/3))


    x = np.array([0, c_1,-c_1, c_2, -c_2, 1,-1])

    #Skalering tilpasset intervallet [a,b]
    x = (b-a)*x/2 + (a+b)/2

    #The corresponding weights
    k_1 = (124+7*np.sqrt(15))/350
    k_2 = (124-7*np.sqrt(15))/350
    w = [256/525, k_1, k_1, k_2, k_2, 1/21, 1/21]

    return np.dot(f(x),w)

if __name__ == "__main__":

    #Feiltoleranse
    tol = 10.0**-10

    #####################################
    #Oppgave 1)

    print('-----------\n')
    print('Oppgave 1 (Sammensatt trapesregel): \n')

    #Feilen
    d = 1

    #Iterasjonen
    n = 2

    #Nyeste og forrige estimat
    new = old = composite_trapezoid(-1,1,f,n)

    #dersom det   velge st rre n ikke gir noe s rlig bedre estimat
    #enn toleransen, er vi ganske n r eksakt l sning
    while d>tol:
        n += 1
        new = composite_trapezoid(-1,1,f,n)
        d = np.abs(new-old)
        old = new
    #Noyaktig svar
    I = np.exp(1.0)-np.exp(-1.0)

    #Konkluderinger
    print("Resultat:",new)
    print("Eksakt svar:",I)
    print("Feil i svaret:",I-new)
    print("n =",n)


    #####################################
    #Oppgave 2)

    print('-----------\n')
    print('Oppgave 2 (Clenshaw-Curtis): \n')

    #Feilen
    d = 1

    #Iterasjonen
    n = 2

    #Nyeste og forrige estimat
    new = old = clenshaw_curtis(f,n)

    #Mens feilen er for stor
    while d>tol:
        n +=2
        new = clenshaw_curtis(f,n)
        d = np.abs(new-old)
        old = new

    #Noyaktig svar
    I_1 = np.exp(1.0)-np.exp(-1.0)

    #Konkluderinger
    print("Resultat:",new)
    print("Eksakt svar:",I_1)
    print("Feil i svaret:",I_1-new)
    print("n =",n)

    #####################################
    #Oppgave 3)

    print('\n----------\n')
    print('Oppgave 3 (Clenshaw-Curtis): \n')

    #Feilen
    d = 1

    #Iterasjonen
    n = 2

    #Nyeste og forrige estimat
    new = old = clenshaw_curtis(g,n)

    #Mens feilen er for stor
    while d>tol:
        n+=2
        new = clenshaw_curtis(g,n)
        d = np.abs(new-old)
        old = new

    #Noyaktig svar
    I_2 = 1.7754996892121809469

    #Konkluderinger
    print("Resultat:",new)
    print("Eksakt svar:",I_2)
    print("Feil i svaret:",I_2-new)
    print("n =",n)

    #####################################
    #Oppgave 4)

    print('\n----------\n')
    print('Oppgave 4 (Gauss-Lobatto): \n')

    #Numerisk integral
    b = gauss_lobatto(f,-1,1)

    #Konkluderinger
    print("Resultat:", b)
    print("Eksakt svar:" , I_1)
    print('Feil i svaret:', b-I_1)

    print('\n----------\n')
```